Mr Long Summary ON ARRAYS

Arrays

- Data structure that contains a collection of the variables (elements) that can store values of the same data type.
- Each unique value is stored at a particular position (indices) and these positions are numbered sequentially.

Declaration of arrays

var <array name> **: array[**LowerIndex..UpperIndex] **of** <data type>

Examples:

- arrGolf : array[1..18] of integer ; **// array of 18 integers**
- arrNames : array[1..4] of string ; **// array of 4 string varaibles**
- arrGrades : array[8..12] of integer ; **// array of 5 integers (first value at position 8)**
- arrTemp : array[-5..5] of real ; **// array of 11 real variables**
- arrSym : array['A'..'F'] of integer ; **// array of 6 integers (positions are characters)**

Can declare multiple arrays of same type:

- **var** *arrPlayer1*, *arrPlayer2*, *arrPlayer3* : array[1..18] of integer ;

Can declare array with values assigned by default

- **var** *arrName* : array[1..4] of **string** = ('James' , 'Sarah' , 'Melanie' , 'Brad');

1	2	3	4
James	Sarah	Melanie	Brad

- **var** *arrNumbers* : array[1..10] of integer = (10, 45, 100, 34, 7, 60, 64, 78, 11, 91);

1	2	3	4	5	6	7	8	9	10
10	45	100	34	7	60	64	78	11	91

Assigning a value to elements in an array

<array name> **[** Position of value (Index) **] :=** <value> ;

Examples:

- arrNames [1] := 'Bruce' ; **// Position 1 of arrNames is assigned value of *'Bruce'***
- arrNumbers [4] := 50 ; **// Position 4 of arrNumbers is assigned value of *50***

Accessing a value in an array

Examples:

- sTemp := arrNames [3] ; **// sTemp is assigned value of Position 3 in arrNames**
- arrNumbers [2] := arrNumbers [2] + 5 ;
  // Increase the value at Position 2 in arrNumbers by 5
- memo1.Lines.add(arrNames [1]) ;
  // Add the value at Position 1 in arrNames to a memo control

For loop to interact with all the values in an array

Examples:

-
  ```
  for K := 1 to 4 do
   memo1.lines.add( arrNames [ K ] ) ;
  ```
  // displays all elements of arrNames in the memo1 control
  (NOTE: *Refer to the looping variable* ***K*** *as the position in the array to LOOP through all values****)***

-
  ```
  for K := 1 to 10 do
    arrNumbers [ K ] := arrNumbers[ K ] + 5 ;
  ```
  // Increase each element in the arrNumbers aray by 5

Size of array integer variable

In many cases an array is declared with a very large upper bound to all for many elements to be stored, however it is possible that the array is *NOT FULL* of elements. An integer variable is declared to store that actual size of the array (the actual number of elements in the array).

Examples:

```
var arrNumbers : array[1..1000] of integer ;        // array can take 1000 integer values
     iSize :  integer ;      // integer stores the ACTUAL number of elements in the array
```

In these cases, you loop from *1* till the *iSize* variable and NOT to 1000.
In case the array does not have values in the array beyond position iSize

-
  ```
  for K := 1 to iSize do
   memo1.lines.add( arrNames [ K ] ) ;
  ```

Aggregate Algorithms with Arrays

- Average of elements in array
- Sum of elements in array
- Count the elements in the array
- Find the maximum (largest) element in the array
- Find the minimum (smallest) element in the array

Average of array – *{Sum and Count included}*

In order to calculate the average, you need to find the sum of all the elements and count how many elements there are.

Sample Code – Average of Array

```
Sum := 0 ;                //initialise Sum (data type same as each array element)
iCount := 0 ;             //initialise iCount variable

for K := 1 to iArraySize do     //1 is lower index, iArraySize is number of values in array
 begin
  Sum := Sum + Array[ K ] ;     //add each element of array to Sum variable
  inc( iCount ) ;               //increase iCount for each element added to Sum
 end; //end of for

rAverage := Sum / iCount ;      // calculate Average AFTER for loop is complete
```

iCount *can be replaced in last line with* ***iArraySize*** *ONLY if you summing ALL the values of the array. The moment you refer to only specific elements in the array (Example: Find average of all values above 50) then you need to keep track of how many of those values exist (using an* ***iCount*** *variable) to correctly calculate the average for that scenario.*

Maximum of array – *{Minimum included}*

Sample Code – Maximum of Array

```
iMax := -999 ;            //initialise iMax to extremely small value - OPPOSITE of what you want
                          //(data type same as each array element)

for K := 1 to iArraySize do     //1 is lower index, iArraySize is number of values in array
 begin
  if Array[ K ] > iMax then     //if element at position K is bigger than our current iMax
  begin
   iMax := Array[ K ] ;         //record our NEW maximum value
   iMaxPosition := K ;          //record the position of NEW maximum value in array
  end; //end of if
 end; //end of for

Showmessage( InttoStr( iMax ) + ‘ is found at position ‘ + Inttostr( iMaxPosition ) ) ;
```

To find the ***minimum*** *(smallest) value in the array, initialise* ***iMin*** *(variable used in place of iMax) to an extremely big value(* ***iMin := 999 ;*** *), and change the* **>** *in the if statement to a* **<**.

Displaying an array

Display ALL the elements of an array in a memo control
(can also be used when adding elements to a rich edit control, list box control or combo box control).

Sample Code – Displaying an array

```
For K := 1 to ArraySize do                //loop through array
   Memo1.lines.add( Array[ K ] ) ;        // array of strings
```

*Use Memo1.Lines.add(**IntToStr** (Array [K])) ; if an array of integers values or **FloatToStr** for array of real values.*

Sorting Arrays

There are many algorithms that can be used to sort an array (put the values in a sequential order, either lowest to highest (numerical values) or from A to Z (text values) or vice versa (descending order). We cover two below:

Bubble Sort

Basic idea of the bubble sort is to loop through an array comparing two adjacent elements and if there are in the wrong order, swop the two elements. Do this repeatedly until no swops are made.
The array is sorted from the back to the front.

Sample Code – Bubble Sort Algorithm

```
iEndCounter := ArraySize – 1 ;         //looping variable for the repeat loop (from the back)
repeat
  bSwapped := false ;         //Indicates that no swapping has taken place for this cycle
  For K := 1 to iEndCounter do
   Begin
        If Array[ K ] > Array[ K + 1 ] then       //compare two adjacent values in array
         Begin
          bSwapped := true ;  // if a swap occurs, set bSwap to true because it is still unsorted
          Temp := Array[ K ] ;          // Temp is same variable type as one element of array
          Array[ K ] := Array[ K + 1 ] ;
          Array[ K + 1 ] := Temp ;
         End ;  //End of IF
   End; //End of L Loop
   Dec( iEndCounter ) ;         //decrease the repeat looping variable
until bSwapped = false ;        //if no swaps occurred, then it must be sorted
```

*The above algorithm sorts the array in ASCENDING order (lowest to biggest). To sort in DESCENDING order change the **>** symbol in the If statement to a **<**.*

Selection Sort

Basic idea of the selection sort is to loop through an array comparing elements in the array until you have the correct element in the first position and continue with the second element until the array is sorted.

The array is sorted from the front to the back.

Sample Code – Selection Sort Algorithm

```
For K := 1 to ArraySize – 1 do         // Loop from 1 to one before the last element
 For L := K + 1 to ArraySize do        //loop from 1 after K's value for rest of the array
  Begin
      If Array[ K ] > Array[ L ] then
       Begin
        Temp := Array[ K ] ;           // Temp is same variable type as one element of array
        Array[ K ] := Array[ L ] ;
        Array[ L ] := Temp ;
       End ;   //End of IF
  End; //End of L Loop
```

The above algorithm sorts the array in ASCENDING order (lowest to biggest). To sort in DESCENDING order change the **>** *symbol in the If statement to a* **<**.

Other sorts include the Merge sort, Quick sort and Insertion sort.

Searching Arrays

Linear Search (unsorted or sorted array)

If there are multiple copies if the same value that you are looking for in an array, then a simply loop from the start to end, checking each value will find all values. If you only want to find the first (or only) occurrence of the value (then stop searching once you have found the value) then make use of the linear search algorithm.

Sample Code – Linear Search Algorithm

```
sInput := InputBox( 'Find', 'What are you looking for:', '' );          // value you are searching for

bFound := FALSE ;      // initialise bFound to false meaning value not found yet
iLoop := 1 ;           //initialise looping variable

while ( bFound = False ) AND ( iLoop <= ArraySize ) do      //Loop while value is NOT found
 Begin                                                      //AND you not at end of array yet
  If Array[ iLoop ] = sInput then       //check element in array with value you looking for
   Begin
     bFound := TRUE ;                   //The value you looking for has been found
     FoundPosition := iLoop ;           //Record it's position in the array so can be used later
   End          //End of If
  ELSE  Inc( iLoop ) ;     //only need to increase looping variable if you haven't found value
  End;          //End of While

If bFound = TRUE then   //if bFound is false, then entire array was search and it never change
 Showmessage(' Found at position ' + Inttostr( FoundPosition ) )    //code if value found
ELSE
 Showmessage(' Not found in array ' ) ;                             //code if value NOT found
```

Binary Search (only on sorted array)

This algorithm only works on a sorted array. It finds the middle value and determines which side would contain the value, then searches that half making use of the midpoint each time to narrow down each segment that is being searched.

Sample Code – Binary Search Algorithm

```
sInput := InputBox( 'Find', 'What are you looking for:', '' );          // value you are searching for

bFound := FALSE ;       // This determines if sInput has been found or not
iLower := 1 ;           //initialise lower limit of section being searched
iUpper := ArraySize ;   //initialise upper limit of section being searched

while ( bFound = False ) AND ( iLower <= iUpper ) do        // Loop while value is NOT found
 Begin                                       //AND Lower and Upper limits have not crossed position
  iMiddle := (iLower + iUpper) DIV 2 ;          //determine midpoint for section being searched
  If Array[ iMiddle ] = sInput then          //if value you looking for at the midpoint of section
   Begin
     bFound := TRUE ;                        //The value you looking for has been found
     FoundPosition := iMiddle ;              //Record it's position in the array so can be used later
   End          //End of If
  ELSE  if sInput > Array[iMiddle] then //value is in right side of midpoint
          iLower := iMiddle + 1
  ELSE    iUpper := iMiddle – 1 ;            //value is in left side of midpoint
  End;          //End of While

If bFound = TRUE then   //if bFound is false, then entire array was search and it never change
 Showmessage(' Found at position ' + Inttostr( FoundPosition ) )     //code if value found
ELSE
 Showmessage(' Not found in array ' ) ;                              //code if value NOT found
```

Parallel Arrays

When you have two or more arrays (with same lower and upper limits) and the same number of elements in the arrays because the value in position 1 of one array corresponds with the value in position 1 of the other array.

- **var** *arrNames* : array[1..4] of **string** ;
  arrNumbers : array[1..4] of integer ;

1	2	3	4
James	Sarah	Melanie	Brad

1	2	3	4
56	78	64	75

The value in position 2 in the ***arrNames*** array (*Sarah*) corresponds with the value in position 2 in the ***arrNumbers*** array (*78*).
(In other words, Sarah's number is 78 or 78 is the number allocated to Sarah)

Rule for parallel arrays is *"what you do to one array, you also do to the other array(s)"*

Example:

- If you find the maximum value in ***arrNumbers*** and record the position, then the person associated with that number is at the recorded position in the ***arrNames*** array.
- If you sort the ***arrNames*** array, then when you swop the position of two values in the ***arrNames*** array, you must also swop the values in the ***arrNumbers*** array at those same positions.

Specific indices

When you are not looping through the array but going to a specific position in the array to edit that value.

Example: ***arrDice*** represents number of times each number on a dice is rolled. Position1 represents how many 1's were rolled, etc.

1	2	3	4	5	6
0	0	0	0	0	0

If a 3 is rolled the just position 3's value is change. Inc(***arrDice*** [3]) ;

1	2	3	4	5	6
0	0	1	0	0	0

Load array from a text file

- Use the reading from text file algorithm
- For each value that is read from text file
    - FIRSTLY, increase the variable storing the size of the array
      number of elements currently in array
    - SECONDLY, place value from text file at position of the size variable
      (mention in step above)
- Remember to initialise the size of array variable to 0 before looping through text file

Sample Code – Loading array from text file

```
If FileExists( 'textfilename.txt' ) = FALSE then      // check if text file exists
 Begin
   Showmessage( 'File not found!' ) ;
   Exit ;
 End;   //End of IF

ArraySize := 0 ;                                      // initialise ArraySize

AssignFile( myFile, 'textfilename.txt' ) ;            // myFile is declared as textfile
                                                      // Example: var F : textfile ;
Reset( myFile ) ;
While NOT eof( myFile ) do
 Begin
  Readln( myFile , sLine ) ;                          //sLine is  a string variable
  {      NEW CODE HERE – adding sLine to Array        }
         Inc( ArraySize ) ;
         Array[ ArraySize ] := sLine ;                //convert sLine if it's Array of integers/Real
  {      END OF NEW CODE    }
  End; //End of while
CloseFile( myFile ) ;
```

*Code in **larger font size** represents NEW array code that is added to reading from a text file algorithm.*

Additional Links:

- Youtube video playlist:
  https://www.youtube.com/watch?v=flcDET8nMpI&list=PLxAS51iVMjv_OTNYfVHF4eu18GwiIw4Q3
- Google drive resource activities:
  https://tinyurl.com/MLE-G11IT-Arrays